Hot Block Clustering for Disk Arrays with Dynamic Striping

= exploitation of access locality and its performance analysis =

Kazuhiko Mogi Masaru Kitsuregawa

Institute of Industrial Science, The University of Tokyo
7-22-1, Roppongi, Minato, Tokyo 106, Japan
{mogi, kitsure}@tkl.iis.u-tokyo.ac.jp

Abstract

RAID5 disk arrays provide high performance and high reliability for reasonable cost. However RAID5 suffers a performance penalty during block updates. In order to overcome this problem, the use of dynamic striping was proposed. This method buffers a number of updates, generates a new stripe composed of newly updated blocks, and then writes the new full stripe back to disks. In this paper, we examine the effect of access locality on the dynamic striping method. To further improve performance in such an environment, we introduce the dynamic clustering policy for hot blocks. Performance analysis with various access localities shows that this method has higher performance than ordinary methods. Performance is also examined for localities that change over time. The dynamic clustering of hot blocks follows locality transitions, showing that under dynamic conditions performance improves.

1 Introduction

Recently, due to the progress of semiconductor technologies, microprocessor performance has improved dramatically while that of secondary storage systems has not kept pace. For secondary storage, the main efforts have been devoted towards increasing capacity and reducing size, with only slight improvements in performance. This has caused the access gap between main memory and secondary storage system to become even larger. Disk array systems have attracted strong attention as high performance secondary storage systems. A RAID5 disk array[8] utilizes a large number of commodity inexpensive drives in parallel to achieve higher performance as well as incorporating parity drives to obtain higher reliability with low storage cost. RAID5 which supports concurrent access of small blocks is currently regarded as one of the most promising approaches for providing highly reliable low cost secondary storage systems.

RAID5 disk arrays employ parity encoding for redundancy. So the new parity for a small write is derived as follows:

$$P_{new} = P_{old} \oplus D_{old} \oplus D_{new} \quad (1)$$

Thus a single block update requires 4 disk accesses: old block read(D_{old}), old parity read(P_{old}), new block write(D_{new}) and new parity write(P_{new}). This deteriorates the throughput of the write operations. In order to overcome this problem and achieve higher performance, several approaches have been proposed, such as head scheduling[10], optimizing the striping unit[2, 3], floating parity/data[6], smart caching[5], parity logging[11], LRAID[1] and dynamic parity grouping[13].

To improve the performance of block updates, we studied storage management methods which use dynamic striping, where instead of updating each block independently, this method buffers a number of updates, generates a new stripe composed of the newly updated blocks, then writes the full stripe onto the free area. We considered two implementations of dynamic striping. One is a LFS[9] based method and the other is Virtual Striping[7]. Both methods which use dynamic striping achieve much higher performance than conventional approaches on randomly issued accesses.

The dynamic striping method needs to make free spaces for dynamic new stripe creations. We call this action garbage collection[1]. To get higher performance

Proceedings of the 21st VLDB Conference
Zürich, Switzerland, 1995

[1] In [9], this action is called segment cleaning. But in Virtual Striping, we don't call a control region a segment thus we use this term.

from dynamic striping we have to reduce the cost of garbage collection. Usually some blocks are frequently accessed (hot blocks) and the others are non-frequently accessed (cold blocks). In [9], the cost-benefit policy, which uses this access locality to efficiently generate free areas, was proposed however only the ratio of extra accesses for the write operations was discussed. The effect of access requests caused by a access locality has not been clarified. This paper examines the effect of access locality for the dynamic striping method.

To make use of the access locality more efficiently for garbage collection with the dynamic striping method, we consider the dynamic clustering of hot blocks, which is an improvement of the cost-benefit policy. We divide each disk into two continuous regions, a hot area and a cold area. All updated blocks are regarded as hot blocks and collected into the hot area. In the hot region, more free space is assigned than in the cold region so that the cost of garbage collection becomes lower than in the case of no separation between hot blocks and cold blocks. Moreover, because frequently accessed blocks area gathered in a limited area, this method decreases the average seek distance, which improves read performance. We examine the feasibility of dynamic clustering of hot blocks on various access localities.

It is possible that the heat of blocks changes randomly. The mechanism of collecting hot blocks follows this change. We also examine performance in the environment of hot spot transition.

In section 2 we survey the methods which have been proposed for improving the performance of RAID5 disk arrays. In section 3 an outline of the storage management schemes using dynamic striping are described. We also show the effect of access locality for these management schemes. In section 4 the details on the method for dynamic clustering of hot blocks are explained. Extensive simulation is done to identify the effect of access localities and the performance characteristics of hot block clustering. Section 5 gives the simulation experiments. Section 6 the results of the analysis.

2 Related works

In this section we review the methods which have been proposed for improving the performance of RAID5 disk arrays. The major problem with using the RAID5 disk arrays is low performance for small write accesses.

As shown in equation (1), traditional methods always perform two pairs of read modify write accesses for each small write access, each requiring rotational delays. To decrease these delays the method named floating parity/data method[6] was proposed. In this method a fixed number of free blocks are reserved in each cylinder. On each update the data block moves from its original location to an appropriate location physically within the same cylinder in order to minimize rotational delays.

Parity logging[11] and LRAID[1] employ different approaches. Both methods delay parity updating by recording update information. When a block is updated, XORed data of the old data and the new data are recorded in the log area. The action of writing to the log area is performed sequentially so that seek and latency delays can be eliminated. In parity logging, log disks and parity disks are distributed over the array, whereas LRAID separates the log and parity disks from the data disks. When the log area becomes full, the controller reads the log and old parities, calculates new parities and writes them out to the proper disks. The accesses for updating the parities require mainly sequential accesses. This type of parity updating requires extra disk accesses, but access efficiency is much higher than traditional update methods. In all, these methods show higher performance than naive RAID5 disk arrays.

Dynamic parity grouping[13] overcomes the parity update penalty by maintaining some parity informations in the controller. It is desirable that blocks which are frequently updated be members of stripes which have their parity values stored in the controller because updating the parities in the controller requires no disk access. But we cannot store all of the parities in the controller, so we need to migrate hot blocks to stripes whose parities are maintained in the controller. The problem becomes how to select blocks which should be allocated to those special stripes. In [13], a sort of access heat is used to guess hot blocks.

To get further performance from RAID5 disk arrays, we have to consider traditional issues such as access scheduling[10], optimizing the striping unit[2, 3], and smart caching[5]. We can combine such methods with the above techniques.

3 Dynamic striping

In naive RAID5 disk arrays, when a block is updated, the new parity has to be recalculated in order to maintain parity consistency, which requires reading the old data block and the old parity block. However if all of the blocks in a stripe are to be updated, then the old data blocks and the parity blocks would not need to be referenced. The new parity can be derived directly from the new blocks. Thus we can avoid the parity generation overhead by dynamic striping method, which allocates a new stripe dynamically for every n block updates, where n is the number of blocks in a stripe excluding parity(figure 1).

This method creates dirty blocks, which have no

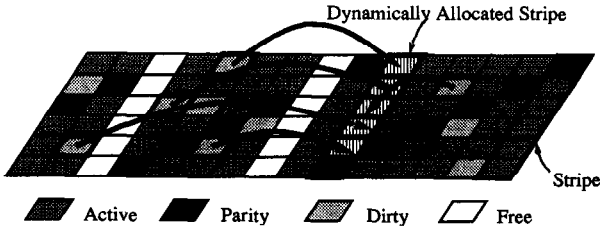


Figure 1: Dynamic striping

valid data because of data updating. Here we assume that free stripes are guaranteed to exist, over which a newly generated stripe can be stored. We cannot write new stripes over dirty blocks because these blocks are still required to maintain parity consistency. To carry out dynamic striping, we must reorganize the parity stripes to produce a free area into which the new parity stripe can be recorded. This reorganization process is called garbage collection. We examined two approaches which employ dynamic striping, with each method adopting a different storage management and garbage collection scheme.

3.1 LFS based storage management (LFS-SM)

The LFS(log-structured file system) was developed to improve the write throughput of the file system. We can apply this storage management scheme to RAID5 disk arrays. The storage management scheme based on LFS is characterized as follows: 1) A set of small write accesses is converted into a large write. 2) Garbage collection is performed by a large management unit (segment).

In LFS-SM, a set of small writes are kept in a non-volatile cache for a while. Once the cache fills to the capacity of a segment, the updated small blocks are written into the free segment. This bulk update reduces write costs. Updated blocks are not written back to their original locations but into a dynamically allocated free segment.

LFS-SM employs segment based garbage collection (called segment cleaning in the original paper[9]). The controller reads all blocks which have valid data (active blocks) in the segment and copies them to the new segment(figure 2). Thus segments which include dirty blocks are cleaned up during segment cleaning.

Cost-benefit policy

The method used to execute segment cleaning must be carefully considered because the efficiency of this action has significant impact on performance. If there is access locality where 90% of the accesses are concentrated on 10% of the data blocks, then about 90%

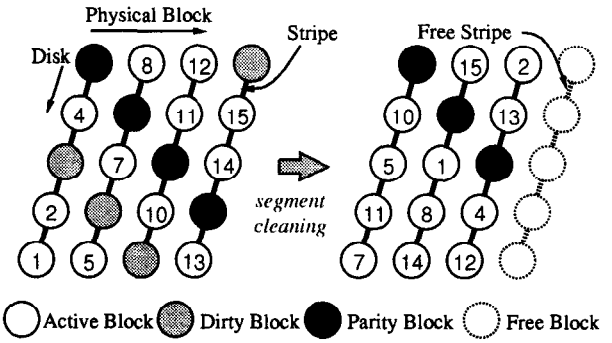


Figure 2: Stripe management scheme based on LFS

of the written blocks are hot blocks. When segments containing a lot of hot blocks are cleaned, the number of active blocks residing in the segment will be low because the number of modified blocks is large. This helps to make the process of segment cleaning quite efficient.

In this scenario, determining the segment to be cleaned is an important factor for the cost of segment cleaning when this action is invoked. In [9], the cost-benefit policy is proposed to get higher efficiency of segment cleaning under access locality. In the cost-benefit policy, the segment to be cleaned is selected by the value of the benefit/cost metric which is calculated as follows.

$$\frac{\text{benefit}}{\text{cost}} = \frac{\text{free space generated} * age}{\text{cost}} = \frac{(1-u) * age}{1+u} \quad (2)$$

In this equation, u denotes the utilization of the segment and *age* denotes the age of the youngest block in the segment. If *age* is small, it is guessed that there are some hot blocks in the segment thus it would be better to wait to clean the segment. In order to efficiently determine which blocks are hot or cold, we need to be careful about distinguishing between write accesses by write requests and those by segment cleaning. Almost all blocks which are written by normal write accesses are hot, and those written back during segment cleaning are considered to be cold.

3.2 Virtual Striping storage management (VS-SM)

In Virtual Striping, parity stripes are virtualized so that we can change the linkage of blocks in a parity stripe dynamically(figure 3). The mapping of the stripe and member blocks is not determined by their physical position but by a table named the Virtual Stripe Table(figure 4) stored in the controller. In the Virtual Stripe Table, the stripe number, the identifiers of blocks in the stripe, and the number of dirty blocks in the stripe are recorded for each stripe. The stripe

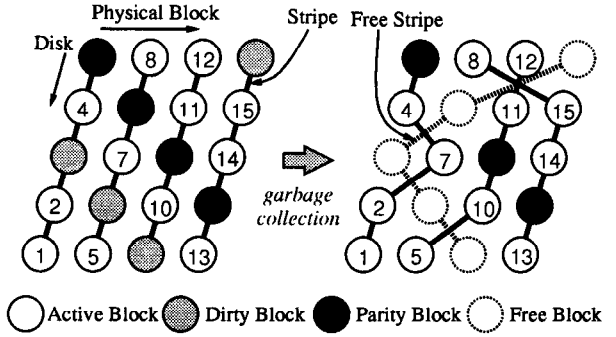


Figure 3: Stripe management scheme with Virtual Striping

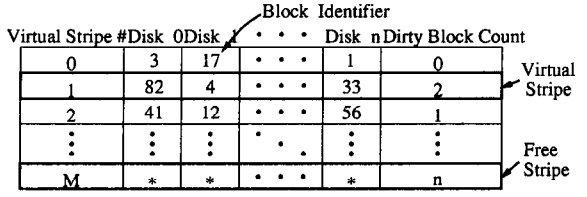


Figure 4: Virtual Stripe Table

number and the block identifiers are used for normal addressing. The dirty block count is used for garbage collection.

In VS-SM, a set of small writes are kept in a non-volatile cache for a while in the same way as in LFS-SM. The threshold value for writing is the size of free blocks in the cylinder which has the largest free area.

Garbage collection in VS-SM is quite different from that of LFS-SM. The garbage collector collects n dirty blocks and creates one free stripe by changing the linkage of stripes as illustrated in figure 3. Garbage collection is performed in the unit of a cylinder because of the access efficiency. The garbage collection process is performed as follows. 1) Determine the target cylinder. 2) Find the base stripe for making a free stripe. We call this stripe the victim stripe. It is desirable that the victim has few active blocks. 3) For each active block in the victim, select a stripe that has a dirty block on the same disk as the active block of the victim. We call this live stripe the partner stripe. 4) Repeat these steps while new victims can be found in the target cylinder. 5) Exchange the active blocks on the victim stripes with the corresponding dirty blocks from the partner stripes. We reference the dirty block count entries in the Virtual Stripe Table to find the victim stripes, and use the Physical Block State Table(figure 5) to determine the state of each block in order to find the partner stripes.

The exchange process requires updating the parity of the partner stripe. Because a new parity is calculated by equation (1), we need 4 accesses to update parity: victim data read(D_{new}), partner data read(D_{old}), partner old parity read(P_{old}) and partner new parity write(P_{new}). Parity update is unnecessary for the victim stripe. The reason why the garbage collector chooses the live stripe with few active blocks is that this reduces the exchange cost.

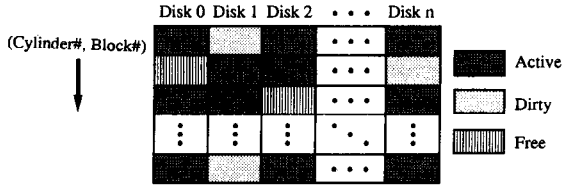


Figure 5: Physical Block State Table

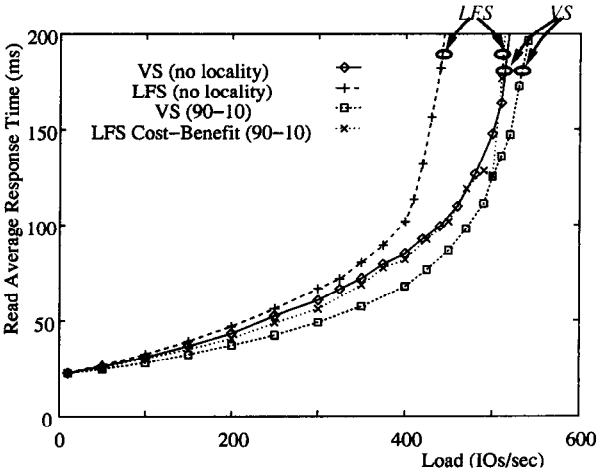


Figure 6: Effect of access locality

3.3 The impact of access locality

In this section we examine the effect of access locality on the performance of the dynamic striping method. We assume that x% of the requests are concentrated to y% of the valid data blocks. We call this access locality "x-y" later. We measured the average read response time with 90-10 access locality and without access locality under the conditions described in section 5. Figure 6 shows the results. In LFS-SM, the cost-benefit policy degrades performance if there is no access locality. Thus we plot the performance without the cost-benefit policy in LFS-SM if there is no access locality.

We can get better performance with 90-10 access locality, which is due to the efficiency of garbage collection. In LFS-SM, the cost of segment cleaning decreases because the cost-benefit policy selects the proper segment for segment cleaning. In VS-SM, the cost of garbage collection decreases because hot blocks are concentrated in the written stripes, this leads to the same effect as explained in the section on the cost-benefit policy (section 3.1).

4 Dynamic clustering of hot blocks

4.1 Separation of the area for hot blocks and the area for cold blocks

Reduction of the cost of garbage collection is the key to high performance in dynamic striping. In general, there are access localities which can be used to improve the performance of RAID5 disk arrays with methods such as caching and dynamic parity grouping. Dynamic striping is also able to use access locality to improve performance with the cost-benefit policy. To gather hot blocks onto a limited area and to increase the amount of recycled free space during garbage collection are the basic ideas for the cost-benefit policy.

Hot segments in LFS are not necessarily clustered together. In this paper, we partition the space physically into two areas, one for hot blocks and one for cold blocks. If there are access localities, it is desirable to collect hot blocks into a continuous area, which concentrates the disk accesses onto a small area, thus decreasing the average head movement distance, which is expected to improve the performance of disk arrays. The dynamic striping method inherently moves the position of blocks when data are updated. Thus the action of collecting hot blocks is quite easily performed.

We manage the two regions independently, that is, writing new data and garbage collection are performed independently on each area. For dynamic clustering of hot blocks, the need to make free space in the hot area is higher than in the cold area because the probability of writing to the hot area is quite large compared to the cold area. It is better to have a larger free space ratio in the hot area than in the cold area, which leads to low garbage collection costs in the hot area.

4.2 The method of clustering hot blocks

We have to be able to distinguish between hot blocks and cold blocks. If the access locality is considered static or predictable, we can use "heat"[12], which is the measure of the number of accesses over a certain period. However, hot blocks may change. In this case, it is difficult to determine an effective method for heat estimation. As described above, almost all blocks written to by normal write accesses can be thought as hot if there are access localities. So we assume that all blocks of normal write accesses are hot in the same way as the cost-benefit policy.

Writing all blocks to the hot area easily consumes the free space of the hot area. When garbage collection is executed on the hot area, we have to move the blocks which are determined to be cold to the cold area. We count the number of blocks which are moved from the cold area to the hot area between two successive garbage collections of the hot area. Then we select the same number of blocks from the hot area as cold blocks moved in and move them from the hot area to the cold area. This block migration is extra work in VS-SM. On the other hand, no extra work is required in LFS-SM, since the garbage collection in LFS-SM executes block migrations.

To determine the cold blocks in the hot area, we use the elapsed time since the last access executed on the block (which excludes the accesses performed by the garbage collector). This method might cause misestimation. Two kinds of incorrect estimation can occur. A hot block is assumed to be cold and a cold block is assumed to be hot. From the performance point of view, the former has a stronger influence than the latter. In order to prevent this phenomena, more space should be allocated to the hot area so that it does not spill out hot blocks.

5 Simulation Experiments

We examine the performance characteristics of hot block clustering through simulation for each of the different storage management schemes (LFS-SM and VS-SM). Optimal organization for each method depends on the workload. Here we assume the loads consist of small accesses, where the average access size is several kilobytes.

Use of cache The use of a write cache is essential for LFS-SM. VS-SM can use the write cache to improve write efficiency. Therefore, we assume the presence of a write cache. In this simulation we employ a cache with a size of 1000 blocks, which corresponds to 1.5 times larger than the number of blocks in a cylinder over 8 disks in the simulation. The cache is a part of the disk controller for both VS-SM and LFS-SM. On the other hand, we don't assume the presence of a read cache because we want to examine the raw performance of the disk array as clearly as possible. We assume that half of the total requests are read operations and the other half are write operations. While the write ratio might look higher than found under normal circumstances, most systems today employ a read cache, thus reads would be absorbed by the cache increasing the ratio of writes accordingly. Since simulation does not employ a read cache, the read/write ratio is set to 50%/50%.

Access scheduling It is very effective to adopt access scheduling because accesses for writing and garbage collection tend to be limited to a very small area. The writing of the dynamic striping method changes the location of updated data to a location determined by the controller. The controller can determine the area for garbage collection in advance. So in most cases there is sufficient time to schedule accesses. This is the reason why we employ access scheduling for the simulation. We use a SCAN based algorithm for

capacity	318MB
cylinders/disk	949
tracks/cylinder	14
sectors/track	6
sector size	4096 bytes
revolution time	13.9ms
seek time model	$\text{seek}(d) = 2.0 + 0.01 \cdot d + 0.46 \cdot \sqrt{d}$
track skew	1 sector

Table 1: Disk model parameters

inter-cylinder and a shortest time first algorithm for intra-cylinder accesses[7].

Execution of garbage collection Garbage collection plays the most important role in dynamic striping. In our experiments with LFS-SM, we invoke garbage collection when the number of free segments becomes less than 6 for the hot area, 4 for the cold area, and 6 if hot block clustering is not employed. For VS-SM, when the number of cylinders which have free stripes becomes less than 9 if hot block clustering is not employed. If it is employed, 6 for the hot area and 4 for the cold area. Successive execution of garbage collection degrades performance. If every disk has some access requests or some garbage collection read requests, we stop making a new garbage collection request. Once such conditions are fulfilled, we determine the target for garbage collection and put its requests onto the proper access queue. Garbage collection is unconditionally performed if there is only one free segment under LFS-SM or there are less than three cylinders which have free stripes under VS-SM.

Other assumptions We make the following assumptions during simulation. 85% of blocks assigned to the data area hold valid data and 15% are used as free space. Access requests are fixed at 4KB. Interval between access request arrivals has a negative exponential distribution. The load is controlled by changing the mean time between access requests. That is, we assume that access requests have a random distribution.

Table 1 shows the disk model parameter, which was employed in [4]. The block size is 4KB. The striping unit is set to the block size. The position of parities is incremented by one track when rotated among the disks. The disk array is composed of 8 data disks and one parity disk (8D+P). In LFS-SM, the segment size is set to half a cylinder. The disk array controller is sufficiently fast so that the overhead of scheduling and table manipulation is negligible. All the control tables are maintained by the controller[2]. After initial

[2]Currently many commercial RAID5 products employ dual controllers as discussed in [5]. Important tables are stored in the NV-RAM of the controllers and consistency between the two copies is maintained through appropriate implementation.

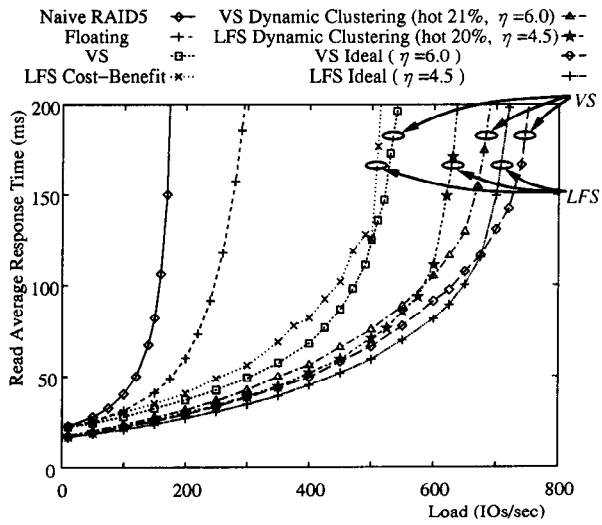


Figure 7: Read response time analysis (90-10 access locality, 8D+P, 85% used)

4 million accesses, statistics collection begins. Thus the active blocks, the dirty blocks, and free blocks are properly distributed over the disks.

6 Evaluations for the dynamic clustering of hot blocks

6.1 Comparison to other methods

Figure 7 shows the average read response time for 100,000 access requests (50,000 read requests) for 90-10 access locality. The horizontal axis shows the mean arrival rates for I/O requests, the vertical axis shows the average read response time. We change the proportion of the hot area and the ratio of free blocks in the hot area. The proportion of hot area is changed by increments of 1% of the total capacity. The ratio of free blocks in the hot area are normalized by the ratio of free blocks in the cold area, because we think that this value corresponds to the ratio of efficiency for garbage collection between the hot area and the cold area. We change the value of η which is calculated according to equation (3) in increments of the value of 0.5 for the variance of free block ratio.

$$\eta = \frac{\dfrac{\text{number of free blocks in the hot area}}{\text{number of blocks in the hot area}}}{\dfrac{\text{number of free blocks in the cold area}}{\text{number of blocks in the cold area}}} \quad (3)$$

In figure 7, we plot the performance lines which were the best. The best performance occurs when the proportion of the hot area is 20% and $\eta = 4.5$ in LFS-SM and the proportion of the hot area is 21% and $\eta = 6.0$ in VS-SM.

In order to compare the performance with that of the other methods, we also plot the performance of naive RAID5 and RAID5 with floating parity/data &

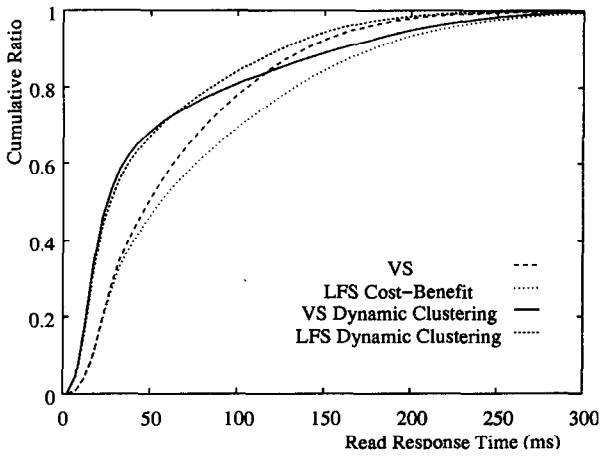


Figure 8: Read response time distribution (90-10 access locality, 400 IOs/sec, 8D+P, 85% used)

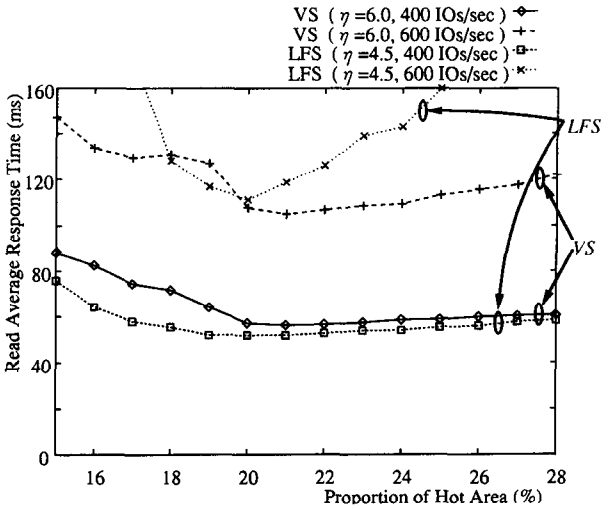
access scheduling in the same figure. We also plot the performance of ideal separation of hot and cold blocks. Here "ideal" means that the controller has the perfect knowledge about the hotness of the blocks.

The storage management schemes which employ dynamic striping have a much larger range of low response times than the other two methods. In the methods using dynamic striping, one with dynamic clustering of hot blocks shows a shorter response time for low loads and can bear higher loads than one which does not use clustering. Hot block clustering shows close to but slightly worse performance than when blocks are ideally separated. At low loads, LFS-SM with clustering shows better performance than VS-SM with clustering. We examine the reason in the next section.

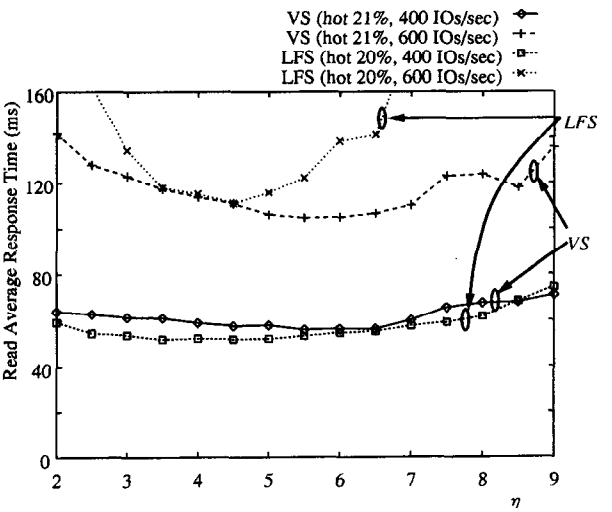
6.2 Read response time distribution analysis

Figure 8 illustrates the distribution for the read response time. These lines are measured through simulation at 400 IOs/sec with 90-10 access locality using the same conditions used for read response time analysis. The horizontal axis shows the read response time. The vertical axis shows the cumulative ratio for the read accesses which have a smaller response time than the given value.

With clustering of hot blocks, the ratio of read accesses which have short response times is higher than without it because a much larger free area is generated by the garbage collection process with clustering than without it and clustering reduces head movements. In VS-SM, there are more accesses which have long response times with clustering than without it. Some read accesses will have to wait for write and garbage collection accesses to finish. The efficiency of garbage collection in the hot area causes the garbage collection and write to take a long time, which makes the read



(a) sensitivity of hot area proportion



(b) sensitivity of η

Figure 9: Sensitivity of parameters (90-10 access locality, 8D+P, 85% used)

response time worse. In LFS-SM, the waiting time for some read accesses is mainly determined by the segment size. In this simulation, the segment size is set to half a cylinder, which is smaller than VS-SM's unit. This is the reason why LFS-SM with dynamic clustering has a shorter response time than VS-SM at low loads.

6.3 Sensitivity of parameters

In this section, we examine the sensitivity of the parameters when we adopt dynamic clustering of hot blocks. We measured the read average access rates at the mean request arrival rate of 400 IOs/sec and 600 IOs/sec with 90-10 access locality for various pa-

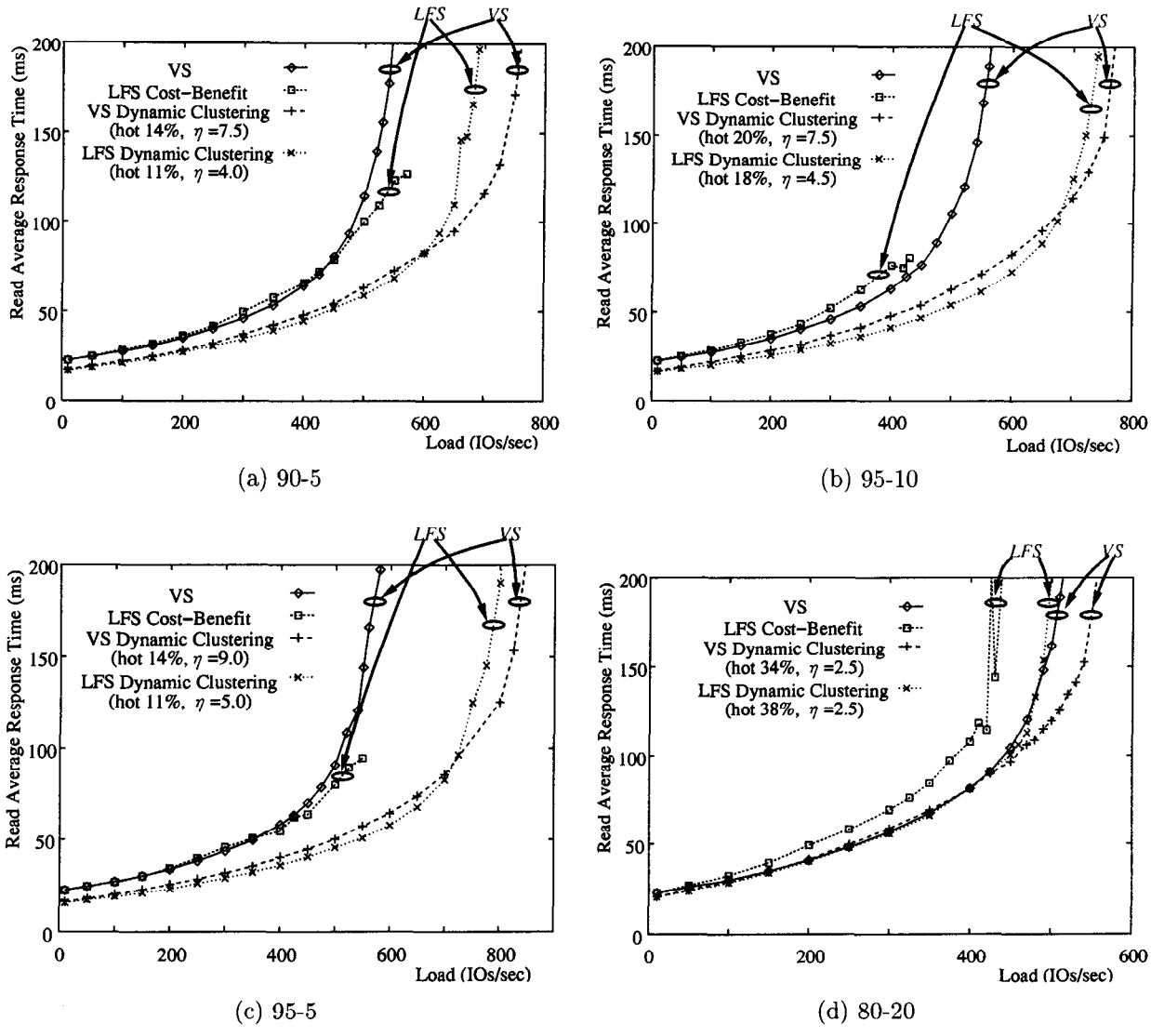


Figure 10: Read response time analysis on various localities (8D+P, 85% used)

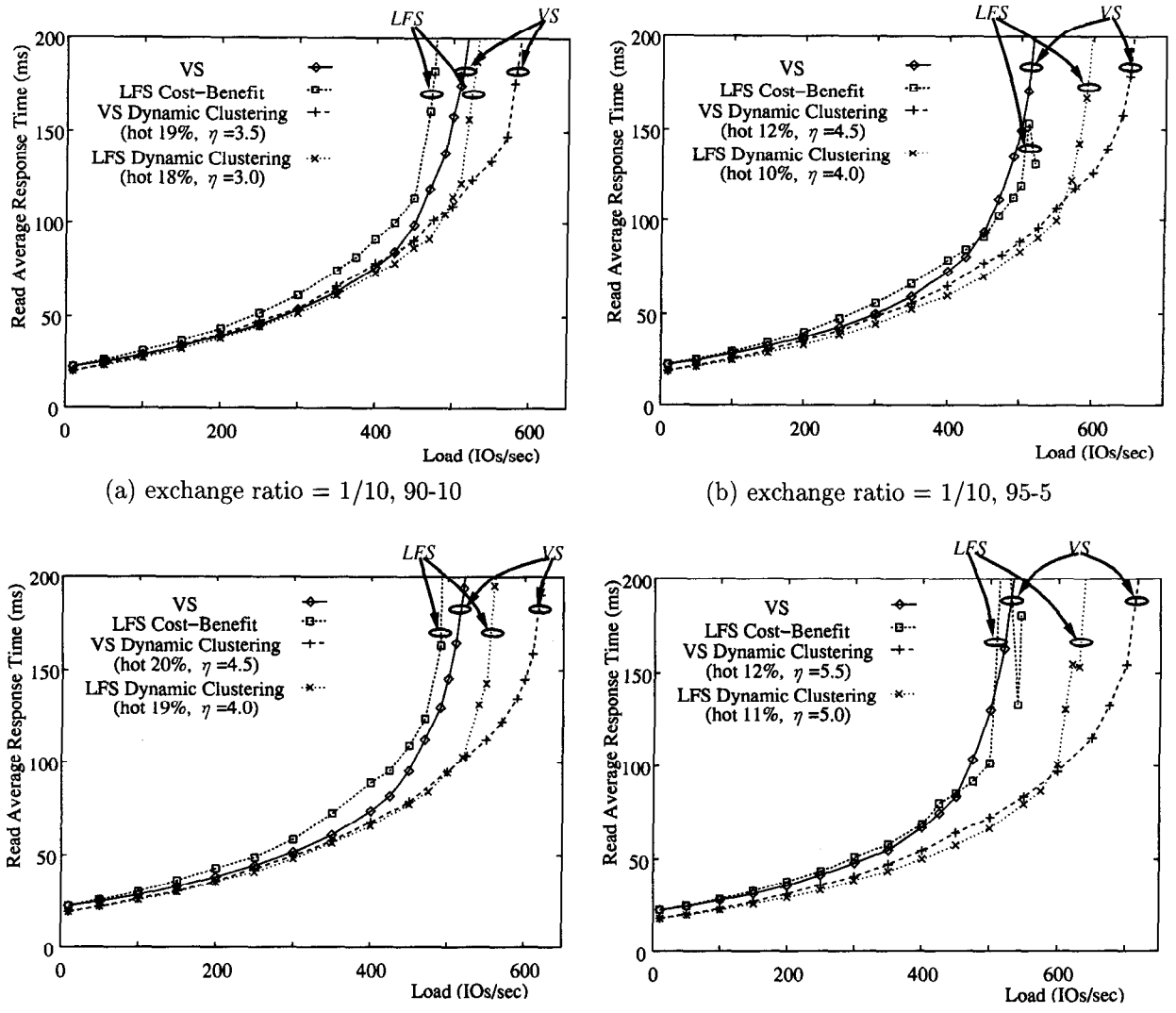
rameters. Figure 9 shows the results. In figure 9 (a), we change the proportion ratio of hot blocks leaving η fixed to the values used in figure 7. In figure 9 (b), the value of η is changed at the fixed proportion ratio of the hot area used in figure 7.

At low loads such as a mean arrival rate of 400 IOs/sec, the response time does not vary for either method. But at a mean arrival rate of 600 IOs/sec at which the best response time is about 100 msec for both methods, the response time is sensitive to the parameters. Under this high load many read accesses must wait until garbage collection accesses or write accesses finish, thus the efficiency of garbage collection significantly affects performance. At high loads, LFS-SM is more sensitive than VS-SM. Although high efficiency has a positive effect, the frequency of garbage collection is also a problem in LFS-SM because the cost of each garbage collection is high. If the proper parameters are set, clustering of hot blocks shows good performance in LFS-SM.

6.4 Variance on the access locality

In this section, we examine the effect of different access localities. We examine the average read response time for various access localities, 90-5, 95-10, 95-5, and 80-20. Figure 10 shows the results. In these figures, we plot two lines for each storage management scheme, one showing the best performance and the other showing the performance without hot block clustering, i.e., normal VS-SM or LFS-SM with cost-benefit policy.

The high access locality improves the performance both with and without hot block clustering. A better improvement is obtained with clustering than without clustering. For low access localities such as 80-20, VS-SM with clustering cannot have a shorter read re-



(a) exchange ratio = 1/10, 90-10

(b) exchange ratio = 1/10, 95-5

(c) exchange ratio = 1/20, 90-10

(d) exchange ratio = 1/20, 95-5

Figure 11: Read response time analysis on locality transitions (8D+P, 85% used)

sponse time at low loads than without clustering because the extra action of cold block migration is expensive in this situation. In LFS-SM, block migrations is a part of garbage collection so we can improve performance at low loads.

6.5 The effect of locality transitions

It is quite possible that the set of hot blocks will change as time passes. In this section, we examine the read response time in an environment of locality transitions. We modeled the transition of locality as follows. The data blocks are divided into two groups, the hot group and the cold group. The number of blocks in the hot group and the access ratio for the hot group is fixed, but the exchange of blocks between the hot group and the cold group are performed randomly. This exchange is executed by one block each. We set the rate of the exchange to 1/10 or 1/20 of the access ratio of hot groups, that is to say, 10 and 20 are selected for the average number of accesses in the hot group. Figure 11 shows the results. In these figures, we plot two lines for each storage management scheme: the best performance with hot block clustering and performance without clustering. For locality transitions, we can improve performance by using dynamic clustering of hot blocks. But the high probability of transitions degrades performance because the transition of hot blocks is equivalent to the cold block writes, i.e., the efficiency of garbage collection is degraded by the transition. The same effect of block migration in VS-SM as explained in section 6.4 is seen at 90-10 locality and a high transition ratio of 1/10.

7 Conclusion

In this paper we examine the impact of access localities on the performance of the dynamic striping method. As far as the authors know, no extensive research has not been done on these issues. Since the access locality is expected to be well exploited for dynamic striping disk arrays, we have throughly performed simulation studies to clarify its viability. It was shown that if the access locality is 90-10, the dynamic striping methods with hot block clustering has much better performance than conventional methods such as naive RAID5 and RAID5 with floating parity/data & access scheduling and better performance than methods which do not use clustering. The clustering of hot blocks decreases the average distance of head movement. The higher ratio of free blocks in the hot area improves the efficiency of garbage collection. These effects lead to a shorter read response time at low loads and more tolerance at high loads.

The higher the access locality becomes, the higher performance becomes. At low locality such as 80-20, VS-SM with clustering cannot improve the performance at low loads because cold block migrations cause high overhead. On the other hand, the clustering of hot blocks improves the performance of LFS-SM because the action of block migrations is combined with garbage collection. We examine the sensitivity of the operating parameters. There is less effect at low loads. But the parameters affect the performance at high loads because the parameters determine the efficiency of garbage collection, which is the main factor of performance at high loads. We can also improve the performance with clustering of hot blocks when the locality changes as time goes on. In this case, the probability of transition affects performance. This behavior was examined.

In this paper, we intended to show the feasibility of using dynamic clustering of hot blocks in various situations. If the access localities are known in advance, we can use the pregiven knowledge to optimize the parameters such as the hot area ratio and the ratio of free blocks in the hot area. The best parameters and performance are somewhat sensitive to access locality, thus the best performance may not be obtained when the actual access locality differs from the predicted. The means of optimizing these parameters for a real environment remains to be investigated.

Acknowledgment

We would like to thank to Stephen Davis for his help to proofreading this paper. This work was supported by a Grant-in-Aid for Scientific Research on Priority Areas #04235103, from the Ministry of Education, Science and Culture, Japan.

References

[1] A. Bhide and D. Dias. RAID Architectures for OLTP. IBM Computer Science Research Report RC 17879 (78489), March 1992.

[2] P. M. Chen and D. A. Patterson. Maximizing Performance in a Striped Disk Array. In *Proc. of 17th Annual Int. Symp. on Computer Architecture(ISCA)*, pp. 322–331, May 1990.

[3] J. Gray, B. Horst, and M. Walker. Parity Striping of Disc Arrays: Low-Cost Reliable Storage with Acceptable Throughput. In *Proc. of 16th VLDB Conf.*, pp. 148–161, August 1990.

[4] M. Holland and G. A. Gibson. Parity Declustering for Continuous Operation in Redundant Disk Arrays. In *Proc. of ASPLOS-V*, pp. 23–34, October 1992.

[5] J. Menon and J. Cortney. The Architecture of a Fault-Tolerant Cached RAID Controller. In *Proc. of 20th ISCA*, pp. 76–86, 1993.

[6] J. Menon and J. Kasson. Methods for Improved Update Performance of Disk Arrays. In *Proc. of 25th Hawaii Int. Conf. on System Science*, vol. I, pp. 74–83, January 1992.

[7] K. Mogi and M. Kitsuregawa. Dynamic Parity Stripe Reorganizations for RAID5 Disk Arrays. In *Proc. of 3rd Int. Conf. on Parallel and Distributed Information Systems*, pp. 17–26, September 1994.

[8] D. A. Patterson, G. A. Gibson, and R. H. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *Proc. of ACM SIGMOD*, pp. 109–116, June 1988.

[9] M. Rosenblum and J. Ousterhout. The Design and Implementation of a Log-Structured File System. In *Proc. of 13th Symp. on Operating Systems Principles*, pp. 1–15, October 1991.

[10] M. I. Seltzer, P. M. Chen, and J. K. Ousterhout. Disk Scheduling Revisited. In *Proc. of USENIX*, pp. 313–323, January 1990.

[11] D. Stodolsky and G. A. Gibson. Parity Logging: Overcoming the Small Write Problem in Redundant Disk Arrays. In *Proc. of 20th ISCA*, pp. 64–75, May 1993.

[12] G. Weikum, P. Zabback, and P. Scheuermann. Dynamic File Allocation in Disk Arrays. In *Proc. of ACM SIGMOD*, pp. 406–415, May 1991.

[13] P. S. Yu, K.-L. Wu, and A. Dan. Dynamic Parity Grouping for Efficient Parity Buffering to Improve Write Performance of RAID-5 Disk Arrays. IBM Computer Science Research Report RC 19041 (83137), July 1993.